University of Hertfordshire
School of Computer Science
BSc Computer Science (Network Protocols and Architectures)

Module: Machine Learning and Neural Computing

Assignment 1

Data Classification Report

Ahmed Aboo
Level 6
Academic Year 2020 - 21

Introduction:

Machine learning is a field of computer science that includes artificial intelligence, statistics, and computer science. Machine learning is concerned with teaching algorithms to recognise patterns and make predictions based on data. Machine learning is particularly useful since it allows us to automate decision-making processes using computers.

Machine learning applications may be found in a variety of places. Nowadays Machine learning is used by big companies like Netflix and Amazon to generate new product suggestions, by banks to detect fraudulent credit card transactions, and healthcare businesses are starting to utilise it to monitor, analyse, and diagnose patients.

In this report we have been given a dataset that has been extracted from the yeast dataset ([1] and [2]). Archive of the original dataset can be found at https://archive-beta.ics.uci.edu/ml/datasets/yeast. This is a dataset including 8 features, used for predicting the cellular localization sites of proteins. The extracted yeast dataset includes 3 classes only. The class is the localization site. Classes 1, 2 and 3 denote CYT (cytosolic or cytoskeletal), NUC (nuclear), MIT (mitochondrial), respectively.

We start off with two data files, one includes the training set, named yeast train.csv, and the other one, yeast test.csv, includes the test data. The last column of each file is the class label indicating the cellular localization sites of proteins.

The training set we are using consists of 800 instances, 288 labelled as 1, 304 labelled as 2 and 208 labelled as 3. For this model, we treated this set as a balanced dataset. The test set contains 335 instances and is of satisfactory quality and requires no pre-processing / data cleansing other than normalisation.

To classify the data, we will be using Support Vector Machines (SVMs). The type of SVM we used was the C-SVC (Cost-Support Vector Classifier) and the kernel function we used was the Gaussian radial basis function (RBF).

The model was written in python 3 using Jupiter notebook and we used functions from the following packages: NumPy, Pandas, Matplotlib, Seaborn and Sklearn.

We then summarised our findings and write conclusions on which model gives a better classification result, if that was what we expected and why.

Data Exploration:

We started off by importing the libraries needed and loading the datasets and exploring their dimensionality, features and their names, labels etc to get a better understanding of the dataset, as can be seen in fig 1.

Fig 1:

Data Classification Assignment

Importing libraries

In [7]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as skl
import time
```

Task 1

Data Exploration

Using (PCA) principle component analysis to understand the characteristics of the dataset.

(a) Import the yeast_train and yeast_test dataset

Use Pandas to load both the training set and the test set. (Let's denote this original training set as training set (I).)

In [8]:
```
train_data = pd.read_csv("yeast_train.csv",)
```

Useing the shape function in pandas to check the dimensionality of the dataset as shown in the following cell.

In [9]:
```
train_data.shape
```

Out[9]:
```
(800, 10)
```

To get a rough idea of this data file's content, you can print the first five using the commands shown in the following cell. You can also print the last five rows with the command data.tail() or even input an integer but (its absolute value should be no more than the total number of rows in the dataset.) in the brackets.

In [10]:
```
train_data.head(10)
```

Out[10]:

	Name	F1	F2	F3	F4	F5	F6	F7	F8	L
0	ADT2_YEAST	0.43	0.67	0.48	0.27	0.5	0.0	0.53	0.22	3
1	ADT3_YEAST	0.64	0.62	0.49	0.15	0.5	0.0	0.53	0.22	3
2	AAR2_YEAST	0.58	0.44	0.57	0.13	0.5	0.0	0.54	0.22	2
3	AATM_YEAST	0.42	0.44	0.48	0.54	0.5	0.0	0.48	0.22	3
4	AATC_YEAST	0.51	0.40	0.56	0.17	0.5	0.5	0.49	0.22	1
5	ABC1_YEAST	0.50	0.54	0.48	0.65	0.5	0.0	0.53	0.22	3

In [16]:
```
test_data = pd.read_csv("yeast_test.csv",)
```

In [17]:
```
test_data.shape
```

Out[17]:
```
(335, 10)
```

In [18]:
```
test_data.head(10)
```

Out[18]:

	Name	F1	F2	F3	F4	F5	F6	F7	F8	L
0	RS6_YEAST	0.46	0.33	0.45	0.22	0.5	0	0.41	0.37	1
1	R19A_YEAST	0.52	0.43	0.61	0.22	0.5	0	0.45	0.22	1
2	R19B_YEAST	0.52	0.43	0.61	0.22	0.5	0	0.45	0.22	1
3	RS41_YEAST	0.32	0.44	0.54	0.21	0.5	0	0.40	0.19	1
4	RS41_YEAST	0.32	0.44	0.54	0.21	0.5	0	0.40	0.19	1
5	RS15_YEAST	0.48	0.41	0.52	0.33	0.5	0	0.51	0.26	1
6	RS22_YEAST	0.46	0.39	0.54	0.22	0.5	0	0.48	0.25	1
7	RS22_YEAST	0.46	0.39	0.54	0.22	0.5	0	0.48	0.25	1
8	RS21_YEAST	0.42	0.50	0.64	0.13	0.5	0	0.49	0.22	1
9	R26A_YEAST	0.46	0.54	0.59	0.41	0.5	0	0.53	0.21	1

In [20]:
```
test_data.columns
```

Out[20]:
```
Index(['Name', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'L'], dtype='object')
```

In [22]:
```
train_data['L'].unique()
```

Out[22]:
```
array([3, 2, 1])
```

In fig 2 we used Principle Component Analysis (PCA) to understand the characteristics of the dataset we started by plotting two subplots in one figure; one was a scatter plot of two features ('F1' and 'F2') of the training set against each other and the other was a scatter plot of the same two features of the test set against each other, normalised the training and test set using StandardScaler and then performed a (PCA) on the scaled training data and plotted the scree plot to report variances captured by each principle component.

Fig 2: Comparing F1 and F2 against each other.

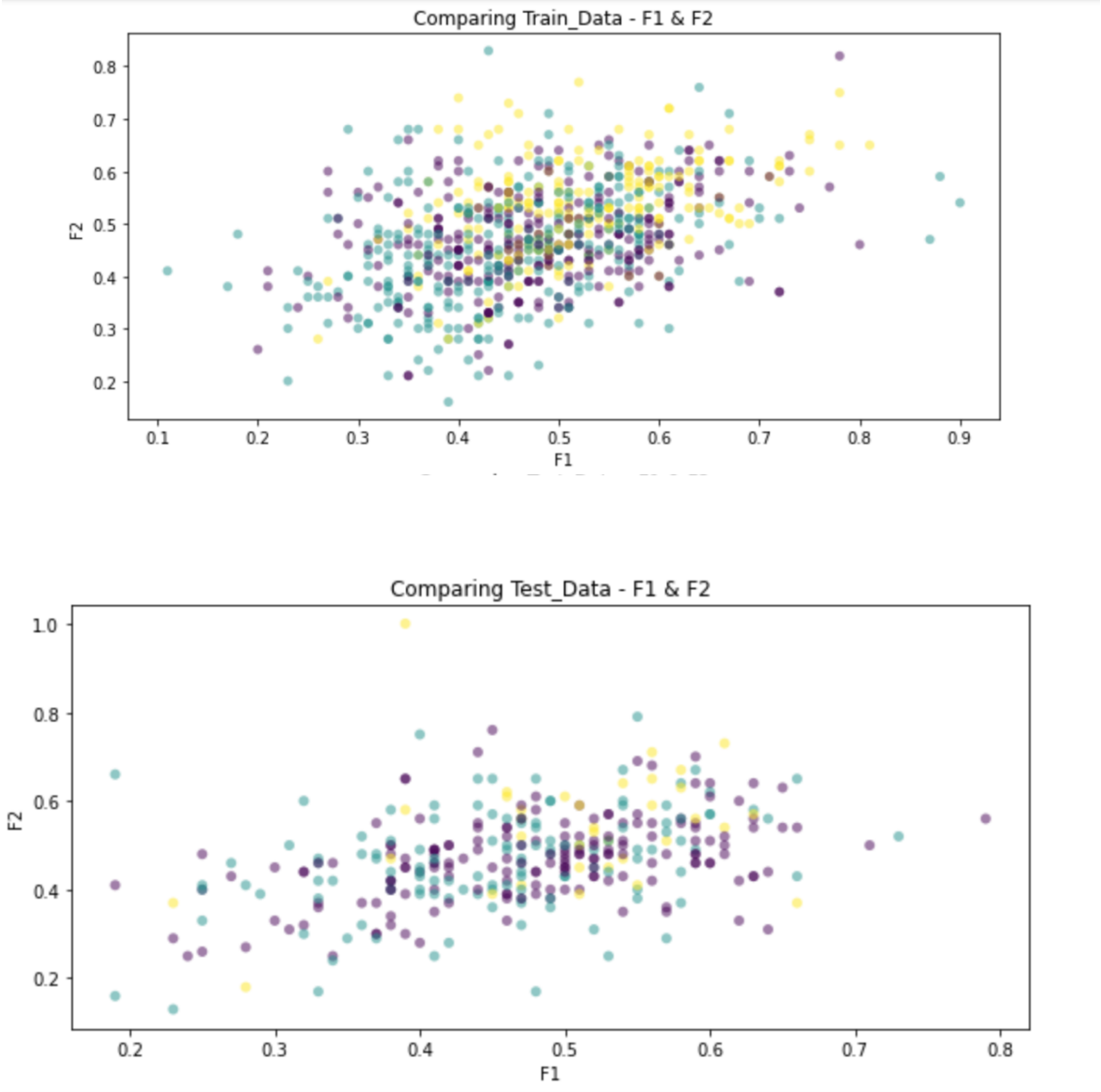


Fig 3: Normalisation using StandardScaler.

(c) Normalisation

Normalise the training set and the test set using StandardScaler() (Hint: the parameters should come from the training set only)

```
: from sklearn.preprocessing import StandardScaler

scaled_train_data = train_data.iloc[:, 1:9]
scaled_test_data = test_data.iloc[:, 1:9]

#Normalise the train and test data using standard scaler.
scaler = StandardScaler().fit(scaled_train_data)
normalised_train_data = scaler.fit_transform(scaled_train_data)
normalised_test_data = scaler.fit_transform(scaled_test_data)

print('='*100)
print('The normalised_train_data.shape is:')
print(normalised_train_data.shape)
print('='*100)
print('The normalised_test_data.shape is:')
print(normalised_test_data.shape)
print('='*100)

#Print the mean value of each feature after removing the mean
print(normalised_train_data.mean(axis=0))
print('='*100)

#Print the standard deviation value of each feature after removing the mean
print(normalised_train_data.std(axis=0))
print('='*100)
```

```
====================================================================================================
The normalised_train_data.shape is:
(800, 8)
====================================================================================================
The normalised_test_data.shape is:
(335, 8)
====================================================================================================
[-6.92779167e-16  2.53130850e-16 -1.06581410e-16 -1.86517468e-16
 -3.75255382e-16 -4.44089210e-18  6.48370246e-16 -5.99520433e-16]
====================================================================================================
[1. 1. 1. 1. 1. 1. 1. 1.]
====================================================================================================
```

We followed by Performing a PCA analysis on the scaled training set and plotted the scree plot to report variances captured by each principal component.

Fig 4:

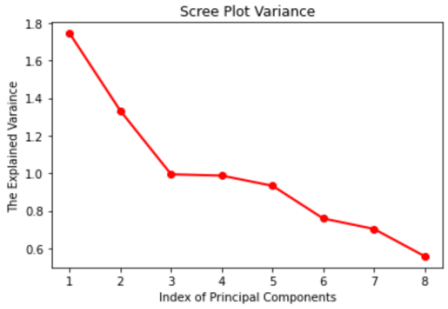
(d) (PCA) Analysis

Perform a PCA analysis on the scaled training set and plot the scree plot to report variances captured by each principal component

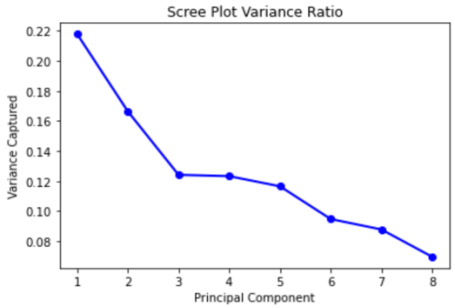
```
: from sklearn.decomposition import PCA

#Initialising a PCA instance.
pca = PCA( )

#The eigen-decomposition is done by using the fit() function; projections of the data in the PCA space is obtained u
pca_data = pca.fit_transform(normalised_train_data)

#Plotting the scree plot.
fig2 = plt.figure()
ax = plt.gca()
plt.plot(np.arange(pca.n_components_) + 1, pca.explained_variance_, 'o-', color='red', linewidth=2,)
ax.set_title("Scree Plot Variance")
ax.set_xlabel("Index of Principal Components")
ax.set_ylabel("The Explained Varaince")

print('='*100)
print('The pca_data shape is:')
print(pca_data.shape);
print('='*100)
print('The Variances Captured are:')
print(pca.explained_variance_)
print('='*100)
print('The Variance Ratio is:')
print(pca.explained_variance_ratio_)
print('='*100)
```

```
====================================================================================================
The pca_data shape is:
(800, 8)
====================================================================================================
The Variances Captured are:
[1.74475835 1.33233648 0.99411061 0.98710851 0.93286719 0.75851931
 0.7028911  0.55742097]
====================================================================================================
The Variance Ratio is:
[0.21782217 0.16633388 0.1241085  0.12323433 0.11646264 0.0946964
 0.08775156 0.06959052]
====================================================================================================
```

Fig 4 Output:




```
[31]: fig3 = plt.figure()
plt.plot(np.arange(pca.n_components_) + 1, pca.explained_variance_ratio_, 'o-', linewidth=2, color='Blue')
plt.title('Scree Plot Variance Ratio')
plt.xlabel('Principal Component')
plt.ylabel('Variance Captured')
plt.show()
```

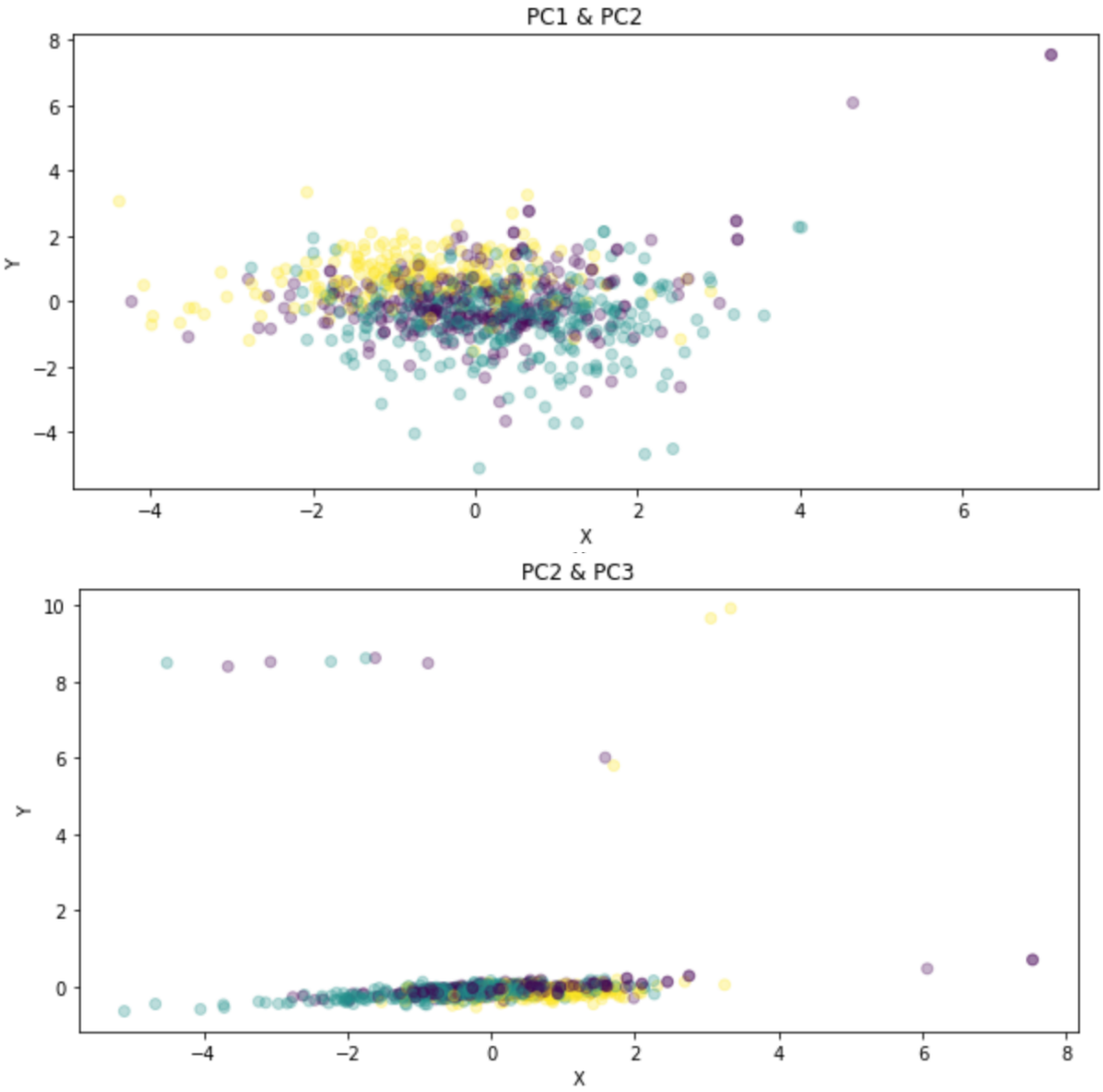


Below in fig 5 we plotted two subplots in one figure; one for projecting the training set in the projection space constructed using the first principal component (PC1) and the second principal component (PC2); the other one for projecting the training set in the projection space constructed using the second principal component (PC2) and the third principal component (PC3).

Fig 5: Plotting Subplots using (PCA) 1 AND (PCA) 2

```
#Plotting 2 subplots in one figure
fig4, ax = plt.subplots(2, figsize=(10, 10))

pca_dataframe = pd.DataFrame(data=pca_data[:,:3],columns=['PC1', 'PC2', 'PC3'])
pca_dataframe = pd.concat([pca_dataframe, train_data["L"]], axis = 1)
pc1_pc2 = pca_dataframe.drop('PC3', axis=1)
pc2_pc3 = pca_dataframe.drop('PC1', axis=1)

#Training data PC1 & PC2
ax[0].set_title("PC1 & PC2")
ax[0].scatter(pc1_pc2["PC1"], pc1_pc2["PC2"], c=pc1_pc2["L"], alpha=0.3)
ax[0].set_xlabel("X")
ax[0].set_ylabel("Y")

#Training data PC2 & PC3
ax[1].scatter(pc2_pc3["PC2"], pc2_pc3["PC3"], c=pc2_pc3["L"],alpha=0.3)
ax[1].set_xlabel("X")
ax[1].set_ylabel("Y")
ax[1].set_title("PC2 & PC3")
```

Fig 5 output:



Once the variance captured was reported and analysed we divided/split the training dataset into a smaller dataset (ii) a validation set, we used between 20% - 30% of the total datapoints using the train_test_split function and reported the number of points in each set, followed by normalisation again with the parameters only coming from the training set as seen in fig 5.

You should always test a classifier on unknown data to determine how well it performs. As a result, divide your data into two pieces before developing a model a training set and a test set.

During the development stage, you utilise the training set to train and assess the model. The trained model is then used to generate predictions on the unknown test set. This method allows you to assess the model's robustness and performance.
Thankfully, sklearn has a method called train test split() that separates your data into these groups. To split the data, first import the function and then use it:

Fig 6:

(b) Normalise both the training set (II) and the validation set

(Hint: the parameters should come from the training set (II) only)

8]:
```
from sklearn import preprocessing

train_data_values = train_data_ii.iloc[:, 1:9]
validation_values = validation_set.iloc[:, 1:9]

validation_labels = validation_set["L"]
training_labels = train_data_ii['L']

#Fit gives mean values and std value bcoz we will use that
#Values for both the sets
scaler = preprocessing.StandardScaler().fit(train_data_values)

"""Scale the smaller training set"""
normalised_train_data_ii = scaler.transform(train_data_values) #First split the smaller training set

"""Scale the validation set"""
normalised_validation_set = scaler.transform(validation_values)

print('='*100)
print('The normalised_train_data_ii.shape is:')
print(normalised_train_data_ii.shape)
print('='*100)
print('The normalised_validation_set.shape is:')
print(normalised_validation_set.shape)
print('='*100)
print(normalised_train_data_ii.mean(axis=0)) #Print the mean value of each feature after removing the mean
print('='*100)
print(normalised_train_data_ii.std(axis=0)) #Print the standard deviation value of each feature after removing the m
print('='*100)
```

```
====================================================================================================
The normalised_train_data_ii.shape is:
(600, 8)
====================================================================================================
The normalised_validation_set.shape is:
(200, 8)
====================================================================================================
[-1.56911521e-16 -2.25005200e-16  3.98940140e-16  1.53950926e-16
  1.50842302e-15 -8.88178420e-18  4.44089210e-16 -7.78636415e-16]
====================================================================================================
[1. 1. 1. 1. 1. 1. 1. 1.]
====================================================================================================
```

We then went on to train an SVM model for each of the 4 different given combinations of cost and gamma, tested it on the normalised validation set and reported the accuracy rate of all combinations.
Followed by selecting the best combination and training another SVM classification model with those selected parameter values, classified the normalised test set this time and reported the accuracy rate and the confusion matrix, seen in fig 7.

Fig 7:

Non Linear Classifications

(a) Basic task

i. Choosing the most suitable parameters When using the C-SVC SVM with the Gaussian radial basis kernel there are two tunable parameters, C (cost) and γ (gamma). You have been given the following combinations: [C=10, γ=0.1], [C=50, γ=0.1], [C=100, γ=0.1], and [C=10, γ=0.01]. You should train an SVM model for each combination from the given 4 combinations and then test it on the normalised validation set. The accuracy rate for each combination on the validation set should be reported. Finally, you need to select the best combination of parameters and report your result.

```
normalised_validation_set.shape
```

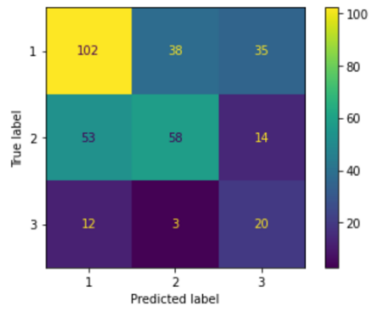
```
(200, 8)
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, ConfusionMatrixDisplay

cost_gamma_combinations = [[10, 0.1],[50, 0.1],[100, 0.1],[10, 0.01]]


for cost, gamma_val in cost_gamma_combinations:
    s = SVC(kernel='rbf', class_weight='balanced', C=cost, gamma=gamma_val,)
    s.fit(normalised_train_data_ii, training_labels)

    train_pred = s.predict(normalised_validation_set)
    print('Accuracy Rate:', accuracy_score(validation_labels, train_pred))

    #print('Classification Report:\n')

    #print(classification_report(validation_labels, train_pred))
```

```
Accuracy Rate: 0.59
Accuracy Rate: 0.55
Accuracy Rate: 0.555
Accuracy Rate: 0.6
```

```
from sklearn.metrics import plot_confusion_matrix

s2 = SVC(kernel='rbf', class_weight='balanced', C=10, gamma=0.01)
s2.fit(normalised_train_data_i, train_labels )

test_labels = test_data['L']

test_pred = s2.predict(normalised_test_data_i,)

print('Accuracy Rate:', accuracy_score(test_labels, test_pred))
plot_confusion_matrix(s2,normalised_test_data_i,test_labels)
plt.show()
print('Classification Report:\n')

print(classification_report(test_labels, test_pred))
```

```
Accuracy Rate: 0.5373134328358209
```




```
Classification Report:

              precision    recall  f1-score   support

           1       0.61      0.58      0.60       175
           2       0.59      0.46      0.52       125
           3       0.29      0.57      0.38        35

    accuracy                           0.54       335
   macro avg       0.50      0.54      0.50       335
weighted avg       0.57      0.54      0.55       335
```

Lastly and in fig 8, we reduced features for the normalised training set and then normalised test set using the first 5 principal components.
Did the classification using the Gaussian radial basis kernel SVM with the best parameter values found from the four different combinations of cost and gamma discussed earlier, trained, and tested the SVM model on the reduced features and reported the classification results on the test set.

Fig 8: Non-linear classification with features reduced using PCA

```
reduced_feature_train_data = normalised_train_data[:, :5]
reduced_feature_test_data = normalised_test_data[:, :5]
```

ii.Do the classification using the Gaussian radial basis kernel SVM with parameter values selected in Task 3 (a) (2 marks). • Normalise the training set and the test set after the feature reduction. • Train an SVM model on the training set with reduced features. • Test the model on the corresponding test set, that is the one with reduced features and report the classification result on the test set.

```
from sklearn.metrics import classification_report

training_labels2 = train_data['L']
test_labels = test_data['L']

scaler_3 = StandardScaler().fit(reduced_feature_train_data)
normalised_train_set_iii = scaler_3.transform(reduced_feature_train_data)
normalised_test_set_iii = scaler_3.transform(reduced_feature_test_data)

s3 = SVC(C=10, gamma=0.01)
s3.fit(normalised_train_set_iii, training_labels2)
y_pred = s3.predict(normalised_test_set_iii)

print(classification_report(test_labels, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.61      0.59      0.60       175
           2       0.50      0.41      0.45       125
           3       0.32      0.57      0.41        35

    accuracy                           0.52       335
   macro avg       0.47      0.52      0.49       335
weighted avg       0.53      0.52      0.52       335
```

Conclusion:

After evaluating both models I believe the model trained on the original features produced a better accuracy result because I think there are some meaningful differences in the that model data compared to the model trained on the reduced data as I think this is because we have more to test on in that model.

I kind of expected this because of the biased split between the training, testing and validation datasets.

To improve our accuracy for the reduced set, I think we should try and perform an arbitrary data split again or maybe use stratified random sampling, and provide more diverse data, or maybe even just use more columns instead of just using five for the reduced dataset and then check for the difference in the accuracy score again.

References:

[1] Nakai, K., and Kanehisa, M. Expert sytem for predicting protein localization sites in gram-negative bacteria. PROTEINS: Structure, Function, and Genetics 11 (1991), 95–110.

[2] Nakai, K., and Kanehisa, M. A knowledge base for predicting protein localization sites in eukaryotic cells. Genomics 14 (1992), 897–911.

[3] Morales, M., 2017. *How To Build a Machine Learning Classifier in Python with Scikit-learn | DigitalOcean*. [online] Digitalocean.com. Available at: <https://www.digitalocean.com/community/tutorials/how-to-build-a-machine-learning-classifier-in-python-with-scikit-learn>.

[4] www.javatpoint.com. 2022. *Support Vector Machine (SVM) Algorithm - Javatpoint*. [online] Available at: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm.